Large-scale analysis of neuroimaging data on commercial clouds with content-aware resource allocation strategies

Large-Scale Analysis of Neuroimaging Data on Commercial Clouds with Content-Aware Resource Allocation Strategies

Massimo Minervini*, Cristian Rusu*, Mario Damiano†, Valter Tucci‡, Angelo Bifone†, Alessandro Gozzi†, Sotirios A. Tsaftaris*,§

*IMT Institute for Advanced Studies, Lucca, Italy
e-mail: {massimo.minervini, cristian.rusu, s.tsaftaris}@imtlucca.it

†Istituto Italiano di Tecnologia, Center for Neuroscience and Cognitive Systems, Rovereto, Italy
e-mail: {mario.damiano, alessandro.gozzi, angelo.bifone}@iit.it

‡Istituto Italiano di Tecnologia, Neuroscience and Brain Technology Department, Genova, Italy
e-mail: valter.tucci@iit.it

§Northwestern University, Department of Electrical Engineering and Computer Science, Evanston, IL, USA

Abstract

The combined use of mice that have genetic mutations (transgenic mouse models) of human pathology and advanced neuroimaging methods (such as MRI) has the potential to radically change how we approach disease understanding, diagnosis and treatment. Morphological changes occurring in the brain of transgenic animals as a result of the interaction between environment and genotype, can be assessed using advanced image analysis methods, an effort described as "mouse brain phenotyping". However, the computational methods involved in the analysis of high-resolution brain images are demanding. While running such analysis on local clusters is possible, not all users have access to such infrastructure and even for those that do, having additional computational capacity can be beneficial (e.g., to meet sudden high throughput demands). In this paper we use a commercial cloud platform for brain neuroimaging and analysis. We achieve a registration-based multi-atlas, multi-template anatomical segmentation, normally a lengthy in time effort, within a few hours. Naturally, performing such analyses on the cloud entails a monetary cost, and it is worthwhile identifying strategies that can allocate resources intelligently. In our context a critical aspect is the identification of how long each job will take. We propose a method that estimates the complexity of an image processing task, a registration, using statistical moments and shape descriptors of the image content. We use this information to learn and predict the completion time of a registration. The proposed approach is easy to deploy, and could serve as an alternative for laboratories that may require instant access to large high performance computing infrastructures. To facilitate adoption from the community we release publicly the source code.

Index Terms

High performance computing, cloud, neuroimaging, MRI, phenotyping, computer vision, image analysis, resource allocation, machine learning.

I. Introduction

The introduction of cloud computing has affected how high performance computing resources are purchased, allocated, and managed. While previously access to such resources required either the purchase of advanced clusters or the participation in grid services, with commercial clouds, super computing level resources are available to everyone. Although initially they were targeted to enterprise users, the use of commercial clouds for research applications is gaining momentum. This transition is expected to facilitate research where large data repositories are needed to be processed ("big data"). An application domain that will benefit from this paradigm change is the large scale analysis of imaging data. Images arising from biological or medical domains (e.g., MRI of the brain) are characterized by high acquisition cost, large resolution, and complex analysis pipelines (i.e., the sequence of image processing steps). However, since the same pipeline is applied for each dataset they are ideally suited for large data parallelization.

The advent of magnetic resonance imaging (MRI) methods for the study of the brain has significantly advanced our understanding of how this important organ functions in health and disease. In typical studies, subjects are divided in two or more groups, based on experimental design. Each subject is then scanned (possibly several times) with a particular scanning protocol to obtain one or several imaging volumes that reflect the underlying anatomy of the brain or brain tissue status or composition. At the next phase, imaging data must be analyzed to extract several features that improve our understanding of the effects of specific genetic variations or disease on the brain.

An example of an analysis is to identify differences in the volume of specific brain regions among the subjects and groups from brain MRI volumes. While an expert can manually delineate these structures in each subject's volume, this process

is highly time consuming (6–10 hours depending on imaging resolution and number of structures) and introduces observer variability. Thus, computational solutions that automatically segment the anatomy within the brain are highly desirable.

One approach that has been particularly popular in the literature is a registration based atlas propagation (Figure 1). Assuming the existence of an atlas $(A_1^{(I)}, A_1^{(L)})$, where $A_1^{(I)}$ is an intensity volume and $A_1^{(L)}$ is the corresponding labeling (i.e., its anatomical segmentation) previously delineated by an expert, the labels can be propagated to another intensity volume $S_1^{(I)}$, if a mapping between the two intensity volumes is known. The result of such a process is the labeled subject $S_1^{(L)}$. An intensity based registration finds a linear or better a non-linear mapping between two volumes using only voxel intensities. Generally, for 1 atlas and N subjects ($S_n^{(I)}$, for $n = 1, \ldots, N$) this process requires N registrations. Several processing platforms implement this approach, e.g., ANTs[1], Freesurfer[2], and LONI[3].

Recently, it has been suggested that is best to use multiple atlases when performing a registration based label propagation [1] to account for variability between the subject population and the atlas. In this context several registrations are performed and the best atlas is identified, usually the one most similar to the subject, to propagate the labels. Alternatively, all atlases are used in a fusion based segmentation approach [1], [2]. In recognition that obtaining multiple atlases is demanding and that atlases may not always reflect population variability, also several intermediate templates are used. These templates are either previous subjects, or even synthetic MRI volumes created by learning the variability between previously processed subject populations [2]. As Figure 2 illustrates, in this case all subjects $S_n^{(I)}$ are registered using multiple templates $T_p = (T_p^{(I)}, T_p^{(L)})$, for $p = 1, \ldots, P$, which are registered to multiple atlases $A_m = (A_m^{(I)}, A_m^{(L)})$, for $m = 1, \ldots, M$, using the intensity component. The final anatomical labels $\tilde{S}_n^{(L)}$ for each subject are obtained by fusing the results of all registration based propagations.

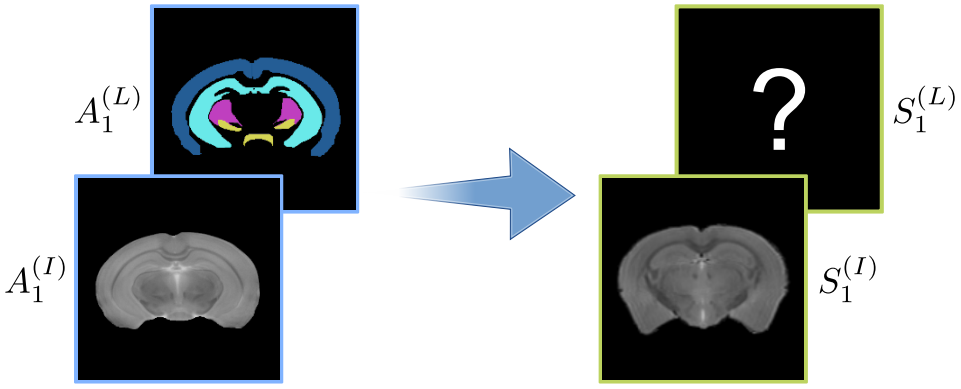


Fig. 1: Registration based propagation of labels from an intensity volume atlas $A_1^{(I)}$ with known anatomical segmentation $A_1^{(L)}$ to a subject volume $S_1^{(I)}$ via intensity based registration to estimate the anatomical labeling $S_1^{(L)}$.

As the number of subjects (N), templates (P), and atlases (M) increases, the complexity increases (in total $P \times (M + N)$ non-linear registrations are required). Depending on the non-linear registration algorithm, the complexity of this process increases even further. Current state-of-the-art algorithms that provide highly accurate, diffeomorphic, and invertible non-linear registrations between volume pairs, are increasingly complex since they optimize models with tens of millions of degrees of freedom while they reach cubic theoretical complexity [3].

It is clear that this analysis is computationally demanding and is just an example of the many complex neuroimaging analysis pipelines used throughout the community [4], [5]. Traditionally, the majority of such analyses has occurred in single powerful workstations with limited throughput. Recently, since non-linear registration of volume pairs (as well as the majority of image analysis pipelines) is suitable to efficient data parallelism, the use of computing clusters to achieve higher throughput has been proposed (see the reviews of [6], [7]).

In recognition that some researchers may not have access to these options, the scientific community reacted by establishing consortia (e.g., LONI, neuGRID, outGRID, CBRAIN) that aim at making distributed (grid) computing available for neuroscience and neuroimaging research [6], [7]. These grid level services usually provide a user-friendly graphical interface for designing and putting together any image analysis pipeline, with respect to both tools and data. However, for executing seamlessly this process, access to the computer grid associated with the consortia underwriting these efforts is assumed.

Unfortunately, users that have limited or no access to powerful computer grids or clusters are left with limited computational options. Even users that do have access to such consortia or local clusters, in some cases they can benefit from having additional computational capacity (for example, increasing throughput to meet a deadline). The users can configure and install

[1]http://stnava.github.io/ANTs/
[2]http://surfer.nmr.mgh.harvard.edu/
[3]http://www.loni.ucla.edu/Software/

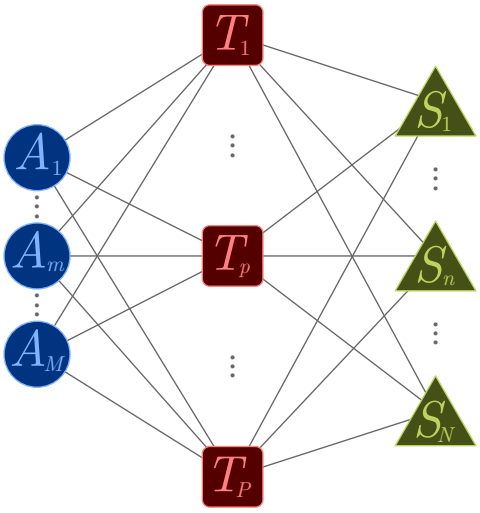


Fig. 2: A multi-atlas (A_m), multi-template (T_p) registration based strategy for anatomical brain segmentation of subject volumes (S_n). The lines represent the warping between two volumes using the deformation fields obtained by non-linear registration.

the underlying computational framework of LONI on another grid environment or the cloud (e.g., they can install the NITRC environment on Amazon EC2[4]); but they have to export the image analysis pipeline as a virtual machine and take care of the details of its execution and the management of the cloud resources. To manage the workflow there exist several systems (Soma-workflow [8], Megha Workflow Management System [9], Gridbus [10], Cloudbus [11], Aneka [12], CometCloud [13]), and several demonstrations of those for imaging applications are available [4], [5]. However, managing such systems (e.g., installation on the cloud environment, and configuration of services) and deployment of jobs may prove challenging and time consuming for most users. Admittedly, these limitations might explain why cloud computing has not been adopted by the broad neuroscience and neuroimaging community for compute-intensive analyses.

In this paper, we use a commercial cloud computing platform (PiCloud) to perform one of such intensive analysis procedures: a registration based multi-atlas, multi-template anatomical segmentation for estimating the size of anatomical brain regions on the basis of magnetic resonance images. Here we show that when PiCloud, which is being developed to facilitate transparent access to commercial cloud infrastructures, is combined with sophisticated analysis and resource optimization methods, it offers an appealing alternative for several laboratories and types of users. Our approach:

- relies on a general purpose cloud environment and not on dedicated grids and clusters;
- utilizes the PiCloud platform, which offers flexible, simple, and transparent allocation of computational resources via a Python application programming interface;
- is based on Python, which is simpler for building complicated image analysis pipelines than other solutions (e.g., using C or C++ programming languages) as others have also found [14];
- is sufficiently easy to be adopted and extended by non computer experts;
- allocates cloud resources efficiently to minimize monetary cost; and
- incorporates a method to learn and predict execution time by taking into account the image content to facilitate resource allocation.

We demonstrate our findings using data from mice phenotyping experiments, with the goal of identifying morphological differences between two different population groups. Our cloud based neuroimaging pipeline identifies such differences in several anatomical parts of the mouse brain. Furthermore, using the same imaging data and execution times we show that it is possible to reduce monetary cost by intelligently allocating resources. This paper builds upon our previous work [15], where we first propose the use of commercial clouds (PiCloud) for neuroimaging.

The rest of this paper is organized as follows: in Section II-A we outline the proposed image analysis pipeline, and in Section II-B we describe the proposed cloud implementation. Section II-C details the proposed framework for intelligent resource allocation on the cloud. Section III outlines implementation details and presents and discusses our experimental findings, while finally Section IV offers conclusions and directions for future work.

[4]http://www.nitrc.org/plugins/mwiki/index.php/nitrc:User_Guide_-_NITRC_Computational_Environment

II. Proposed Methods

A. Registration Based Brain Anatomical Segmentation Pipeline

To demonstrate the potential of high throughput processing on the cloud we implement a typical example of a compute-intensive analysis: a multi-atlas, multi-template anatomical segmentation such as the one illustrated in Figure 2. We rely on a highly accurate non-linear registration algorithm to match a moving volume (e.g., a subject $S_n^{(I)}$) and a fixed volume (e.g., a template $T_p^{(I)}$). This registration finds a diffeomorphic transformation that is symmetric, invertible, topology preserving, and differentiable [3], [16]. This allows us to propagate information from the fixed to the moving volume and vice-versa in an accurate manner. Briefly described, the goal of non-linear registration is to find the optimal transformation, ϕ, within a specified transformation space which maps each location $\mathbf{x}$ of volume $\mathcal{S}(\mathbf{x})$ to a location in volume $\mathcal{T}(\mathbf{z})$ such that a specified cost function relating the similarity between $\mathcal{S}$ and $\mathcal{T}$, is minimized. Recently, diffeomorphisms (ϕ) have been proposed that are differentiable, invertible, and topology preserving. With diffeomorphic registration, the following variational function is optimized:

$$\{\boldsymbol{v}_1^*, \boldsymbol{v}_2^*\} = \underset{\boldsymbol{v}_{1,2}}{\operatorname{argmin}} \left\{ \int_0^{0.5} ||L\boldsymbol{v}_1(x,t)||^2 \, dt + \int_0^{0.5} ||L\boldsymbol{v}_2(x,t)||^2 \, dt + \lambda \int_\Omega \Pi_\sim \left(\mathcal{S} \circ \phi_1(\mathbf{x}, 0.5), \mathcal{T} \circ \phi_2(\mathbf{x}, 0.5)\right) d\Omega \right\} \quad (1)$$

where, $t \in [0, 0.5]$, $\boldsymbol{v}(x,t) = \boldsymbol{v}_1(x,t)$ and $\boldsymbol{v}(x,t) = \boldsymbol{v}_2(x, 1-t)$ when $t \in [0.5, 1]$, $\Pi_\sim$ is a volume similarity metric, L is an appropriate norm, λ is a regularizer, and Ω is the domain of the diffeomorphism. While finding the optimal transform is complex and ill-defined, under certain assumptions, symmetric diffeomorphic solutions are obtained numerically using greedy optimization strategies [16]. To increase the probability that the greedy strategy finds a global minimum and to reduce the number of necessary iterations, a multi-resolution strategy using up to LV levels of the Gaussian pyramid is adopted. A transformation is found first for an under-sampled version of the original data, and then is refined at each step while reaching the final full resolution.

The results of each registration are linear (affine) registration parameters as well as forward (subject to template, or template to atlas) and inverse (template to subject, or atlas to template) warp fields (essentially voxel-to-voxel mappings) [16]. Using the inverse warp field and affine parameters obtained we estimate $S_n^{(L)}$ using the given labeled template $T_p^{(L)}$, denoted by $S_n^{(L)}|T_p^{(L)}$. The same process can be applied to obtain $T_p^{(L)}|A_m^{(L)}$. Technically speaking we do not perform these steps independently, i.e., first propagating from atlas to template and then from template to subject, but we concatenate the transforms to minimize interpolation errors.

Finally, we combine all intermediate segmentations ($S_n^{(L)}|T_1^{(L)}, \ldots, S_n^{(L)}|T_P^{(L)}$) by a fusion process to obtain the final result $\tilde{S}_n^{(L)} = \bigcup_{p=1}^{P} S_n^{(L)}|T_p^{(L)}$, implemented as a majority vote algorithm that maintains for each voxel the most populous label across the P candidates [1].

B. Proposed Cloud Implementation

To translate our image analysis pipeline to the cloud we use the PiCloud[5] platform. This platform provides the user with transparent access to cloud resources currently supplied by Amazon Web Services (AWS). A Python library and middleware (implemented as an API) enables the use of cloud storage and computational power directly via the Python interpreter. It relieves the user from dealing with several issues such as virtual servers, connections, resource scaling, job scheduling, and billing, which may prevent non computer experts from using cloud computing services.

Among the types of service offered by PiCloud (as of July 2013), we consider the following:

- c1 core type, offering 1 compute unit, 300 MB of memory, and low I/O performance at $0.05/hour;
- c2 core type, offering 2.5 compute units, 800 MB of memory, and moderate I/O performance at $0.13/hour;
- f2 core type, offering 5.5 compute units, 3.7 GB of memory, and moderate I/O performance at $0.22/hour;
- m1 core type, offering 3.25 compute units, 8 GB of memory, and high I/O performance at $0.30/hour;

where core types are similar to what Amazon refers to as instances, a compute unit refers to AWS definition of equivalent CPU capacity of a 1.0–1.2 GHz 2007 Opteron or 2007 Xeon processor, and I/O performance refers to internal communication within the cloud infrastructure. Each core type can be used in two modes: on demand or real-time core provisioning. PiCloud charges on a per millisecond rate and not for the full hour when using on demand cores.

The PiCloud service can provide a virtually unlimited number of cores, hence not imposing any strict limits on the total number of potential jobs parallelized simultaneously. This, coupled with the fact that the jobs are self-contained, translates into a high speedup of the overall process.

Subject to resource availability and internal proprietary resource management, any job executed on demand on PiCloud is placed on a queue, incurring a variable (unknown to the user) waiting time. Allocating a priori a job for real-time operation

[5] http://www.multyvac.com/

guarantees immediate execution, with a fixed waiting time. If the usage meets a certain per hour minimum ($> 60\%$ of the hour), the real-time allocation is free of charge. Data storage and download incur a cost, whereas data uploading and intra-cloud communications (between storage and node) are free of charge.

We configure a Ubuntu GNU/Linux virtual machine (actually a Linux Container) following the PiCloud instructions for setting an environment to include customized libraries and applications required for our operation. We optimize (by enabling multi-threading) and compile the ANTs toolkit, which offers registration (`ANTS`), warping (`WarpImageMultiTransform`), and label fusion (`ImageMath`), and other standalone applications. Furthermore, NeuroDebian (http://neuro.debian.net/) repositories have also been included to facilitate installation of other neuroimaging applications [6].

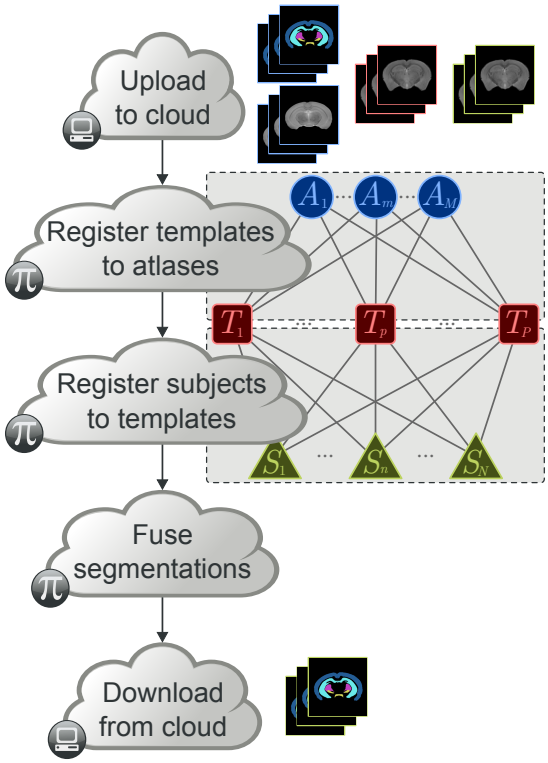


Fig. 3: Schematic of the workflow that performs the strategy of Figure 2 for a given set of data (subjects, templates, and atlases).

We use Python scripts to execute the workflow illustrated in Figure 3 for a given set of input volumes (subjects, templates, and atlases). More specifically,

(a) We use the PiCloud file storage interface, in particular the `cloud.files.put` function, to upload the input data (i.e., subjects, templates, and atlases) to the cloud from our local workstation.
(b) We use Python wrapper functions, an abstract example of which is shown in Listing 1, that execute the components of the image analysis pipeline for each input dataset: they retrieve the input data (subjects, templates, or atlases) from cloud storage to a node, run the applications (registration or fusion) composing the pipeline as subprocesses, and finally save the output files on cloud storage.
(c) A single Python instruction of the PiCloud library (`cloud.map`) executes (maps) the functions for each dataset pair (e.g., an atlas and a template, or a template and a subject), specifying the type of computational resources (core type, on demand or real time) and custom environment to use. This function essentially implements the core of the data parallelism that we employ here.
(d) We retrieve the output data (e.g., labeled subjects $\tilde{S}_n^{(L)}$) from the cloud, using the `cloud.files.get` function of the PiCloud file storage interface.

Once all jobs have completed (registrations and fusions), we download to our local workstation all final labeled volumes ($\tilde{S}_n^{(L)}$) for subsequent statistical analysis. The web interface and the APIs available by PiCloud, provide several statistics (execution times), metadata, and output logs on each job, which are collected for analysis and comparisons.

To execute jobs on the PiCloud system (via the `cloud.map` arguments) we must identify the type of service (real-time or on demand) and the core type (c1, c2, f2, or m1) among the function's arguments. Each option involves different charges and maintains different guarantees for service. One can assign all jobs to the same service type for simplicity however this is far from

Listing 1 Example abstract code illustrating a part of the proposed cloud implementation, discussed in Section II-B.

```
import cloud
import subprocess

def job(moving, fixed, params):
    # retrieve input volumes
    cloud.files.get(moving + '.nii.gz')
    cloud.files.get(fixed + '.nii.gz')
    # parse parameters and input files
    build_ants_arguments = '... params ...'
    args = 'ANTS' + build_ants_arguments
    popen = subprocess.Popen(args, shell=True, stderr=subprocess.STDOUT)
    popen.wait() # wait for ANTs to terminate
    cloud.files.put('...') # save output files
```

an optimal allocation. While for certain cases assigning the type of core is trivial (e.g., the c1 does not have enough memory to execute a registration), when several core types satisfy the requirements a resource allocation methodology is necessary. Several approaches exist to assign resources (and schedule them if needed [17], [18]) by satisfying various performance criteria, such as (1) cumulative, makespan, or minimum execution time or cost or deadline [19]–[23]; (2) constrained formulations such as minimizing cost under time constraints [24], [25]; and (3) allocating resources under budget constraints [26]. Nevertheless, all options essentially care for the proper assignment of each job to a core and service type.[6]

More critically, most approaches do require an estimate of the execution time for each job and application to make efficient and accurate allocation and scheduling decisions [5], [19]. Typically these estimates are obtained by analytical modeling of the underlying source code (which is not always available), or using empirical and historical data of each job and application type [5], [19], [28], [29]. However, when the job to be executed (in our case a registration) is homogeneous (i.e., the same job and parameters is executed on different data of the same size), we would have to rely on an average estimate of execution time [30]–[32]. Unfortunately, this option underperforms when the execution time depends on the actual data content (not only their size) and involves non-linear iterative components. Image registration is a perfect example of a highly non-linear process that iterates until a numerical minimum is found and thus is highly dependent on the input volumes. In fact, the expected convergence time varies largely among volume pairs [5].

Thus, in order to intelligently allocate resources in an environment that is composed of similar jobs but the data content changes, an approach that utilizes this data dependence to more accurately estimate execution time is necessary.

C. Predicting the Execution Time of Image Analysis Applications

Here we propose for the first time a learning based method, which utilizes prior history of execution times and actual imaging data to learn a model that relates image content and execution time for each individual registration. Although we focus on registration, our formulation is general enough to accommodate several image analysis applications that involve iterative optimization components. It may be utilized wherever estimates of execution time are needed (e.g., in any resource management framework [5], [19]).

To showcase the effect of accurate prediction of execution time from a resource management perspective we consider a proof of concept problem: minimizing the total cost of performing a multi-atlas, multi-template segmentation for a set of subjects, templates, and atlases. Under certain assumptions (e.g., ignoring total completion time, or makespan, and not incorporating the unknown to the user wait time for on demand cores) this optimization reduces to minimizing the individual cost of each registration. The main innovation is the method to predict accurately the execution time of non-linear processes. Once a good estimate of time is available it can be incorporated into any complex optimization and resource scheduling process.

For convenience of presentation and without loss of generality, we refer to a registration between a subject $S_n^{(I)}$ (i.e., the moving volume) and a template $T_p^{(I)}$ (i.e., the fixed volume), and assume that all volumes have the same dimensions and data size (as such we ignore time for internal to the cloud data transfers, i.e., application time $\simeq$ execution time). We denote as $f_\$$ the function that maps the execution time $t_{n,p}^c$ of a pair of volumes $S_n^{(I)}$, $T_p^{(I)}$ to monetary cost for a given core type c (c1, c2, f2, or m1). Under our assumptions, the minimum total cost is achieved by minimizing the cost of each individual registration

[6]Note that we do not refer here to cost optimization of renting services from a mix of providers [27].

and as such we are interested in which core type c minimizes:

$$c^*_{n,p} = \underset{c}{\operatorname{argmin}} f_{\$}(t^c_{n,p}), \tag{2}$$

where $c^*_{n,p}$ identifies the minimum cost allocation for a particular volume pair.

Unfortunately, the execution time of an application is not known a priori and a model to estimate it is necessary. We model the execution time $t^c_{n,p}$ of a pair of volumes $S^{(I)}_n$, $T^{(I)}_p$ on core type c, as:

$$t^c_{n,p} = \frac{f_{\text{app}}}{f^c_{\text{CPU}}}, \tag{3}$$

where f^c_{CPU} reflects the underlying CPU capabilities of core type c, while f_{app} is a function that reflects how many processing cycles the application requires. Although in Eq. (3) a simple linear relation is adopted, more complex models that relate application cycles to processing time exist; however, the following analysis still holds and would only require that another Eq. (3) is adopted.

Most works in the literature related to scheduling or resource provisioning would assume either the worst case scenario for the number of cycles needed for an application (by source code profiling) [32], or they would rely on a historical average estimate [30], [31], or learned models of the usage of computational resources for an application [28], [29]. For the same application (e.g., a registration) with data size equal for each case, this means that the execution time is either estimated using a constant (a worst case value is assumed) or as the historical mean (or maximum) of previous executions for the same core type. However, in this context this will not lead to good allocation, because it does not take into account the effect of the actual data content on the execution time of the application (e.g., how many iterations does it take to converge).

We propose that in the context of non-linear image registration, we can actually model the application component more accurately by incorporating the image content within the formulation as:

$$f_{\text{app}} = f_{\text{param}}(\boldsymbol{\tau}) \cdot f_{\text{cont}}(S^{(I)}_n, T^{(I)}_p, \boldsymbol{\tau}), \tag{4}$$

where f_{param} is a multiplicative component that relates how different registration parameter options (represented as a set $\boldsymbol{\tau}$) of the application affect the required number of cycles per iteration; while f_{cont} represents how many iterations are necessary to achieve the minimum in the non-linear optimization function of the transformation model given parameters $\boldsymbol{\tau}$. Substituting Eq. (4) to Eq. (3) we obtain:

$$t^c_{n,p} = \frac{f_{\text{param}}(\boldsymbol{\tau})}{f^c_{\text{CPU}}} \cdot f_{\text{cont}}(S^{(I)}_n, T^{(I)}_p, \boldsymbol{\tau}). \tag{5}$$

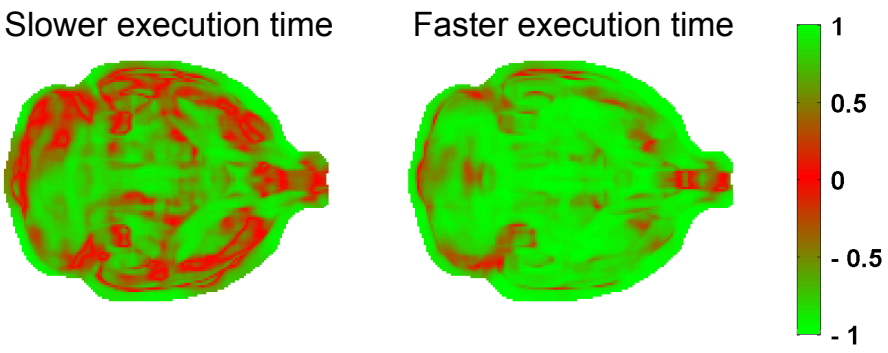
For simplicity but without loss of generality, we can assume that all jobs are executed with the same parameters:

$$t^c_{n,p} = \text{const}^c \cdot f_{\text{cont}}(S^{(I)}_n, T^{(I)}_p, \boldsymbol{\tau}). \tag{6}$$

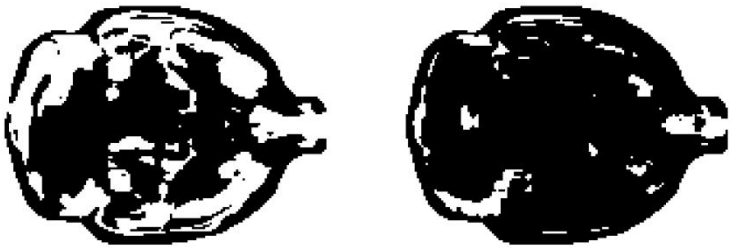
It is known that intensity based registration algorithms utilize local similarity metrics, such as normalized cross-correlation, mean squared error, and normalized mutual information, to drive the registration process [16]. Thus, the similarity of two volumes should affect how fast and easy will be to register them. Figure 4a provides a motivating example, showing color coded local correlation values (using a $w \times w \times w$ sliding window, with $w = 5$) for two different volume pairs at full resolution: one with slow execution time and one faster. We see that the slower pair has more areas with low similarity (colored as red hues) when compared to the faster one. As commonly done in neuroimaging, here we assume that the two volumes are first registered linearly to the same common space and as such gross volumetric differences are non existent, and only local variation must be accounted for.

Based on this motivation, we proceed by modeling $f_{\text{cont}}(S^{(I)}_n, T^{(I)}_p, \boldsymbol{\tau})$ to reflect the similarity between $S^{(I)}_n$, $T^{(I)}_p$. We must pick a function that is computationally easy to estimate (in other words it does not take long to compute), but should reflect the diversity of volume pairs a registration algorithm encounters. It should also relate to the similarity criteria the registration algorithm used (hence our inclusion of the parameter set $\boldsymbol{\tau}$ to reflect the type of similarity metric and the neighborhood size utilized in the registration). Using, for example, the average across all slices of local similarities should be a good indicator of similarity, but is not the only one. In fact, the statistical distribution and the topology of local similarities convey important information as Figure 4b illustrates. Therefore, we propose to use statistical features that characterize the distribution of similarity values and shape descriptors to approximate the topological distribution of high similarity values between two volumes.

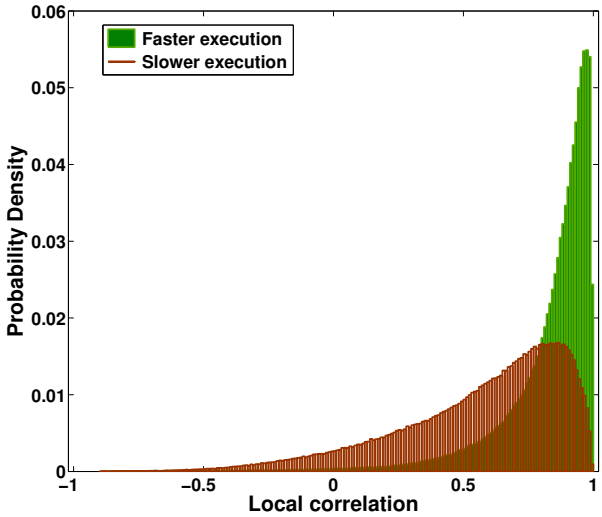
In order to reflect the contribution to time of processing the original volumes hierarchically, using LV levels of the Gaussian pyramid obtained by subsampling the original resolution, we represent as $\mathcal{V}^g_{n,p}$ the three dimensional matrix of the collected values of similarity between two volumes $S^{(I)}_n$, $T^{(I)}_p$ at the g^{th} level of the Gaussian pyramid obtained using a local similarity criterion in a $w \times w \times w$ neighborhood as defined in the set $\boldsymbol{\tau}$.



(a)



(b)



(c)

Fig. 4: Distribution of local correlation is related to execution time. (a) The local correlation values vary spatially in this slice view between different volume pairs $\mathcal{V}_1$, $\mathcal{V}_2$: one with slower execution time (left) and one with faster (right). (b) Sliced volumes $|\mathcal{V}_1|_h$, $|\mathcal{V}_2|_h$ with threshold $h = 0.5$ used to compute the topological features. For these volumes, the values of the topological features are: Sp = (480.5, 42.7), Cd = (0.64, 0.67) and Fl = (0.67, 0.93) for the pairs of slower and faster execution time respectively. (c) The probability density of local correlation values of slower or faster volume pairs ($\mathcal{V}_1$ and $\mathcal{V}_2$) are different and their characteristics are used to predict execution time.

On the basis of this volume $\mathcal{V}^g_{n,p}$ we extract the mean, standard deviation, kurtosis, skewness, and median of the vector $\boldsymbol{\rho}^g_{n,p}$, which is the collection of local correlation values of the volume $\mathcal{V}^g_{n,p}$.

The above statistical features do not take into account the topological distribution of correlated areas. Thus, we also adopt several, easy to compute, topological 3D shape descriptors introduced in the discrete space (a voxel representation) of binary volumes. We obtain the binary volume $|\mathcal{V}^g_{n,p}|_h$ after thresholding with value h the absolute value volume $|\mathcal{V}^g_{n,p}|$, such that an empty voxel (0) means low similarity and a filled voxel (1) means high similarity. We adopt several descriptors, such as sphericity (Sp) [33], discrete compactness (Cd) [34], and fill (Fl) that are computationally efficient. Using these topological features we aim to capture the compactness and density of the similarity volume, since it is known that recovering several small deformations that are spaced far apart requires more effort from a registration standpoint than a larger but co-located

and compact deformation.

Sphericity Sp [33] is defined as:

$$\mathrm{Sp} = \frac{A^3}{36\pi V^2}, \tag{7}$$

where A is the area of the enclosing surface and V is the volume of the space. Considering every measurement to be unit length, computing V reduces to counting the filled voxels in the space while A corresponds to the sum of the areas of the external polygons (1 in each case) of the voxels which are on the visible faces of $\mathcal{V}^g_{n,p}$. Sphericity achieves its minimal value of 1 for a perfect sphere shape. When comparing sphericity of multiple objects, the values are scaled to produce a meaningful, comparable result in the range $[0, 1]$.

Discrete compactness Cd [34] is computed as:

$$\mathrm{Cd} = \frac{A_c}{A_{c_{\max}}}, \tag{8}$$

where A_c is the contact surface area which corresponds to the sum of the areas of the contact surfaces (1 in each case) common to each two voxels, while $A_{c_{\max}} = 3(n - n^{2/3})$ for a n-voxel representation. Cd is in the range $[0, 1]$, with 0 achieved when all voxels are disconnected and 1 for a perfect cube shape.

Fill Fl is a basic measure defined as:

$$\mathrm{Fl} = \frac{V}{V_{\text{total}}}, \tag{9}$$

where V_{total} counts the total voxels in the object representation. Fl is again in the range $[0, 1]$.

For each new registration we compute the local correlation volume $\mathcal{V}^g_{n,p}$ (and $|\mathcal{V}^g_{n,p}|_h$) and extract the previously defined features (a process which is extremely fast compared to registration). Based on such features and a model of execution time, which we learn as we will describe subsequently, we estimate the time to perform the registration, in order to identify the core allocation that minimizes monetary cost in our PiCloud implementation. For a collection of new registrations we split the jobs accordingly, by passing the respective volume pairs and core types as `cloud.map` arguments.

We use supervised learning approaches to obtain models of the execution time $t^c_{n,p}$ of intensity based volume registration from historical data. Given a collection of ℓ previously processed data samples (input volume pairs, and time) on a particular computational set-up c, we represent the local similarity features (extracted at each of the LV levels of the Gaussian pyramid) as a matrix $\mathbf{X} \in \mathbb{R}^{\ell \times d}$, where $d = 8 \cdot LV$ due to the 5 statistical and the 3 topological features at each of the LV levels.

We linearize Eq. (6) by taking the logarithm on each side, therefore we proceed by modeling $\log(t^c_{n,p})$ as the dependent variable in all of the regression formulations that follow. Accordingly, a predicted execution time $\log(\hat{t}^c_{n,p})$ is transformed back to the original scale (seconds), prior to using its value to calculate the monetary cost of executing the corresponding registration on core type c. To predict execution time of non-linear registration, we adopt a variety of linear and non-linear regression models from the machine learning literature (i.e., linear regression, Random Forest [35], and support vector regression [36], [37]), which we briefly review in the following paragraphs.

The first type of regression analysis we consider is a simple linear model, which expresses execution time (actually, the logarithm of time) as linear combination of the input features:

$$y_i = \beta_0 + \beta_1 x_{i,1} + \cdots + \beta_d x_{i,d}, \tag{10}$$

where $y_i \in \mathbb{R}$ is the response (i.e., the collection of logarithmic scaled execution times), $\beta_0 \in \mathbb{R}$ is the intercept, $\beta_1, \ldots, \beta_d \in \mathbb{R}$ are the regression coefficients, and $x_{i,j} \in \mathbb{R}$ are the predictors, with $i = 1, \ldots, \ell$ and $j = 1, \ldots, d$. We estimate the model parameters that identify the regression hyperplane using a least squares formulation. With the coefficients $\hat{\beta}_0, \ldots, \hat{\beta}_d$ now known, we predict the computational time for an unseen volume pair using Eq. (10). We should note that in Eq. (10), if we ignore the contribution of the content (in other words assume $\beta_1 = \ldots = \beta_d = 0$), solving the least squares problem to find the optimal constant is equivalent to using the historical mean of execution time at each core type as a predictor of future execution time.

To see if some features (in their extracted level) are more important than others, we optimize the number of features employed by the linear model using a feature selection strategy widely adopted for sparse approximations, in order to discard unnecessary predictors and obtain a possibly more interpretable model [38]. We explicitly impose the maximum allowed representation error $\varepsilon \in \mathbb{R}$, with respect to the model obtained using the full support of predictors (i.e., all features are used). We adopt a greedy method that gradually reduces the support of the solution by iteratively removing the predictors that offer the smallest model error increase, while remaining below the target imposed representation error ε [39].

In addition to the linear model, we also investigate possible non-linear relations between execution time and features, using random forest regression [35] and support vector regression [36], [37]. Random Forest regression [35] uses an ensemble of n_{tree} unpruned regression trees $\{h(\mathbf{x}, \Theta_k),\, k = 1, \ldots, n_{\text{tree}}\}$, where $\mathbf{x}$ is a d-dimensional feature vector, and Θ_k are random vectors determining bootstrap samples of the original dataset. When growing a tree, the best split at each node is selected

among a random subset of $m_{\text{try}} \ll d$ features. A new data sample $\mathbf{x}'$ is evaluated at each tree, whose output is the numerical value corresponding to the terminal node reached by applying the set of rules for that tree. The prediction of the random forest for $\mathbf{x}'$ is the average across all individual tree predictions. Since features are selected randomly when building the trees, random forest regression performs feature selection implicitly.

On the other hand, from the area of maximum margin classifiers, Support Vector Regression (SVR) [36] is a statistical learning method for the computation of a regression function from labeled training data. We first use the basic linear formulation of SVR, that operates in the original d-dimensional input space, thus restricting the regression function to be a hyperplane. We also investigate the non-linear fit to the training data, using a function $\varphi : \mathbb{R}^d \to \mathcal{H}$ to map the input patterns $\mathbf{x}_i \in \mathbb{R}^d$ into a Hilbert space $\mathcal{H}$, possibly higher dimensional, where the best approximating function could be linear. Support vector regression is formulated as a convex optimization problem [36]:

$$\begin{aligned} \underset{\mathbf{w},\xi,\xi^*,b}{\text{minimize}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{\ell}(\xi_i + \xi_i^*) \\ \text{subject to} \quad & \mathbf{w}^\top\varphi(\mathbf{x}_i) + b - y_i \leq \epsilon + \xi_i, \\ & y_i - \mathbf{w}^\top\varphi(\mathbf{x}_i) - b \leq \epsilon + \xi_i^*, \\ & \xi_i, \xi_i^* \geq 0, \end{aligned} \tag{11}$$

where ℓ is the number of training samples, $\mathbf{w}$ and b are the coefficients of the hyperplane, $C > 0$ is a regularization parameter controlling the flatness of the regression function, ϵ determines the width of the insensitivity zone (i.e., the margin), ξ_i, ξ_i^* are the slack variables, y_i is the target value, and $i = 1, \ldots, \ell$. The optimization problem of Eq. (11) is solved in its dual form using the Lagrange multipliers method, to identify the support vectors that define the regression function in the input space. In general, evaluating φ explicitly could be costly (or even unfeasible), hence the mapping is computed efficiently using a kernel function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, such that $k(\mathbf{x}_i, \mathbf{x}_j) = \langle\varphi(\mathbf{x}_i), \varphi(\mathbf{x}_j)\rangle_{\mathcal{H}}$, where $\langle\cdot,\cdot\rangle_{\mathcal{H}}$ denotes the inner product in the high dimensional space. We include in our study the following kernels widely adopted in the statistical learning theory:

1) Linear Kernel: $k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^\top \mathbf{x}_j$,
2) Gaussian Kernel: $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2)$,
3) Sigmoid Kernel: $k(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma\mathbf{x}_i^\top \mathbf{x}_j + r)$, and
4) Polynomial Kernel: $k(\mathbf{x}_i, \mathbf{x}_j) = (\gamma\mathbf{x}_i^\top \mathbf{x}_j + r)^a$.

The parameters of the SVR learning algorithm (C, ϵ) and kernel functions (γ, r, a) are selected using a grid search strategy.

We use different regression models in order to cover maximally the advantages offered by each one in recovering/interpreting the underlying data structure. For example, the Linear Model albeit simple is generally able to produce good, interpretable results – especially with the addition of a feature selection step – and can be easily extended to process data in an online fashion, thus updating the model on the fly with every new addition of data. On the other hand, SVR methods are able to accommodate for non-linear dependencies, even though the generated models rarely have a real-world interpretation and a considerable amount of time has to be allocated to the delicate task of parameter tuning. Finally, Random Forest also allows for non-linearity and additionally can accommodate categorical variables in the model (e.g., if we were to use one model to describe all possible parameter and environment setups) while also providing a natural feature selection mechanism (variable importance estimation). Again, the selection of the model may be time consuming since many realizations of the Random Forest are computed for the same parameters and the best model is kept.

In the following section we present the performance of the proposed prediction approaches compared to the classical one of using the historical mean of execution time.

III. Results and Discussion

A. Experimental Setup

To demonstrate the potential of our approach, in this paper we use imaging data from small animal (mice) phenotyping experiments. We use total $N = 20$ subject volumes, $P = 6$ template volumes (from other mice experiments), and $M = 1$ atlas (from [40]). While all the mice are of the same age, ten mice are common wild-types and ten are transgenic individuals carrying a mutation thought to mimic genetic variation related to a rare human disease. The subjects are scanned ex-vivo with a 7T Bruker Scanner, using a T2-weighted RARE sequence at a 0.09 mm isotropic resolution, resulting in volumes of $177 \times 200 \times 177$ voxels. All volumes are first processed to isolate the brain (skull stripping) and to correct for intensity inhomogeneity using standard approaches. Our aim is to identify and quantitate the presence of brain morphological abnormalities in transgenic individuals.

Parameters for all ANTs based registrations are normalized cross correlation as similarity criterion in a window of size $w = 5$ voxels (i.e., $5 \times 5 \times 5$ neighborhood), $LV = 4$, a maximum of 100 non-linear iterations at each level, and Gaussian regularization and symmetric normalization parameters as recommended in [16].

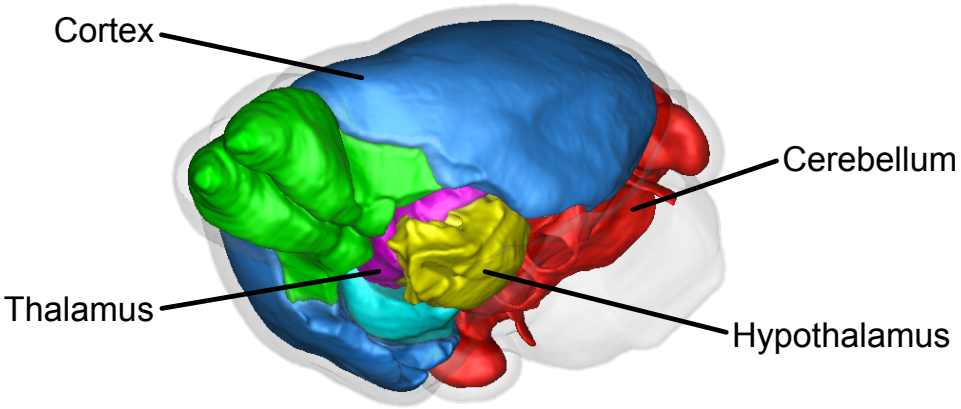


Fig. 5: A 3D rendering of brain anatomy for one of our subjects. Four different anatomical structures are labeled and shown with different colors, while the brain volume is reported as semi-transparent gray.

We execute the image analysis pipeline using the proposed PiCloud implementation only on 'real-time' cores. Thus, 6 jobs (registration between atlas and templates) are required for the first part of the analysis, and 120 jobs (registration between templates and subjects) for the second part. Also, 20 fusion jobs are necessary. Each job is assigned to a core instance on the PiCloud, with locks to complete the registration processes (126 jobs) before launching the fusion jobs. The benefit of using 'real-time' cores is that each instance (and job) is launched simultaneously after a fixed provisioning time.

To compare the gain in efficiency by parallelizing this process using the PiCloud platform, we also execute on a local workstation (8 core 3.4 GHz Intel Core i7 and 16 GB RAM) the same process using 1 (serially) or 8 cores (parallel) without any optimization.

B. Phenotyping Findings

Figure 5 shows a 3D rendering of four structures of interest for one of our subjects. Collectively Table I shows average (and standard deviation) volume sizes for the four areas of interest, which are obtained by measuring the number of voxels that have a given label number in the labeled subject volume $\tilde{S}_n^{(L)}$. Observe that the transgenic mice exhibit larger cerebellar and thalamic volumes, with respect to the wild type group. However, total intracranial volume (TIV) is shown to be statistically different among the two groups, where the transgenic mice exhibit larger brain sizes. To identify differences in relative volume, we obtain normalized structure volumes, where the size of each structure is normalized by the TIV of that subject. It is evident now that overall the size for all of these four structures is altered in the transgenic group, indicating gross morphometric differences in the brain structure of this transgenic mouse line. (All statistical findings are obtained using Mann-Whitney-Wilcoxon non parametric tests.)

TABLE I: Morphological phenotyping results for the two mouse lines used where structure volumes are shown as mean (standard deviation). *, *** indicate significant difference at the 0.05, 0.005 levels, respectively.

	Volume (mm^3)		Normalized Volume (%)	
Structure	Wild Type	Transgenic	Wild Type	Transgenic
Cortex	67.2(2.85)	67.44(5.31)	13.73(0.18)	12.85(0.5)***
Cerebellum	57.7(4.22)	44.83(5.08)***	11.78(0.54)	8.53(0.58)***
Thalamus	14.99(0.54)	17.09(1.27)***	3.06(0.09)	3.26(0.19)*
Hypothalamus	8.27(0.3)	8.43(0.47)	1.7(0.06)	1.61(0.05)***
Total Intracranial Volume (TIV)	489.41(17.6)	524.53(31.06)*	-	-

C. Computational Results

In this section we present our findings related to the computational aspects of the process as well as the results of the learning based modeling of the execution time. All models use the previously defined probability densities and the topological features of the local correlation volumes at all $LV = 4$ levels.

As depicted in Figure 3, the workflow assumes that the user uploads the input data to the cloud and, upon completion, downloads the output data. We measure the upload and download rate from our site in Italy to PiCloud (hosted in East Coast

TABLE II: Average times and cost per non-linear registration on cloud and local workstation shown as mean (standard deviation).

Resource Type	Average Time (min)		Average Cost[b] ($)
	Data Transfer[a]	Processing	
PiCloud c2	0.27 (0.17)	58.28 (22.83)	0.13 (0.05)
PiCloud m1	0.19 (0.13)	43.83 (15.16)	0.22 (0.08)
PiCloud f2	0.19 (0.02)	37.23 (10.44)	0.14 (0.04)
Workstation	–	21.47 (7.07)	–

[a] This accounts for time spent moving around data within the PiCloud infrastructure, reflecting both network level I/O latency and also latency to the local (node) storage. For reference, input data size = 5040 ± 136 kB, output data size = 44605 ± 362 kB.
[b] Costs according to PiCloud pricing of July 2013.

AWS) at 3.7 and 7 Mbit/s respectively. Assuming on average 3 MB for each volume, the total times spent uploading and downloading data are 2.9 minutes and 69 seconds respectively, since we upload 27 volumes and download only 20 volumes (the final labeled subjects). Average times (for intra-cloud data transfer and processing time) for each non-linear registration as well as monetary cost are shown in Table II. Observe that with respect to processing time, data transfer time is negligible, demonstrating the compute-intensive characteristics of this analysis, making it ideal for data parallelism, which is known to scale efficiently.

The makespan of the overall process (in our context the execution time of the longest registration job for the first part of the analysis, followed by the longest registration for the second part, and finally the longest fusion job) for our study on PiCloud requires 8, 5.1, and 4.2 hours for c2, m1, and f2 core types, respectively. On the other hand using all 8 cores on the local workstation requires a total of 18 hours. The total time at the workstation without any parallelization (1 core) requires over 140 hours (approximately 6 days). As the number of registrations or the time to perform a registration increase (higher resolution, different parameters) the local workstation is not able to handle the load.

Considering a hypothetical experiment (a similar one is described in [2]) with $M = 5$ atlases, $P = 200$ synthetic templates, and $N = 45$ subjects, will require $P \times (M+N) = 10000$ registrations. A workstation with 8 cores, would require approximately 57 days. Even assuming the existence of a cluster/grid service that would guarantee 32 simultaneous jobs, this hypothetical experiment would need 14 days. On the other hand, PiCloud offers high scalability by allowing the user to provision the required resources for real-time service; as such from a computation view point the makespan of this experiment will be similar to doing a small experiment. Naturally the only limiting factors may be the time spent uploading and downloading data to and from the cloud (insignificant in our case), the guarantee of service up-time, and the accumulated total cost. It is therefore important to identify strategies that may help mitigate the cost involved.

To showcase the potential of the proposed learning framework for predicting execution time of non-linear registration (for minimizing cost), we use real execution times and the 120 volume pairs of the second stage of the registration process (the right hand part of the graph in Figure 2). These registrations are executed in all possible core services (c2, f2, m1) and thus we collect 360 execution times in total. For reference, the total cost of executing all 120 registrations on c2, f2, and m1 is $17.2, $18.1, and $30.1, respectively. In the following we refer only to c2 and f2, because m1 is more expensive than f2 and slower in all cases. Since for each registration we know the actual execution time at c2 or f2 core service, using Eq. (2) we identify the optimal scenario of assigning a registration to the core type that minimizes cost. We label this assignment as $c^*_{n,p}$ and the actual time as $t^c_{n,p}$. This mix of c2 and f2 core services is referred to as the *best case* scenario in the following.

Subsequently we use the volume pairs and execution times for each core service to train all the models described in Section II-C. We use the same similarity criterion (correlation) and the same neighborhood size (cube of 5 voxels) as those used for ANTs registrations to obtain $\mathcal{V}^g_{n,p}$ and extract the statistical and topological features from this correlation map. We use a Matlab implementation of Random Forest[7] and built-in functions for linear regression. For support vector regression we use LIBSVM (version 3.17) [41].

To test the accuracy of predicting the execution time of an unseen volume pair $S^{(I)}_{\bar{n}}, T^{(I)}_{\bar{p}}$ we adopt an unbiased cross validation strategy, resembling the leave one out validation approach. We take the original dataset consisting of 120 registrations (volume pairs) and their execution times at each core type, and for each volume pair we perform a round of cross validation, using that pair for testing and a subset of the remaining for training. To ensure that the models are trained without encountering any of the testing volumes, and hence are blinded to the testing data, we identify which volumes are involved (for example, $S^{(I)}_1$ and $T^{(I)}_2$). To assure that no bias exists, we eliminate from the dataset all pairs that involve in a registration any of the two volumes (for our example all registrations that involve either $S^{(I)}_1$ or $T^{(I)}_2$ are excluded). We use the remaining volume and time pairs (called here the *unbiased training set*) to train the models described in Section II-C, using parameters reported

[7] http://code.google.com/p/randomforest-matlab/

TABLE III: Regression model parameters used in the experiments.

Model	Parameters
Linear Model	–
Linear Model + Feature Selection	$\varepsilon = 16\%$
Random Forest	$n_{\text{tree}} = 90$, $m_{\text{try}} = 6$
SVR + Linear Kernel	$C = 6$, $\epsilon = 0.1$
SVR + Gaussian Kernel	$C = 2$, $\epsilon = 0.1$, $\gamma = 0.1$

TABLE IV: Cross-validated results of regression analysis for c2 and f2 core type execution times, reported as Mean Absolute Percentage Error (MAPE).

Model	Training		Testing	
	c2 (%)	f2 (%)	c2 (%)	f2 (%)
Proposed				
Linear Model	13.0	11.2	17.6	15.5
Linear Model + Feature Selection	14.3	12.3	24.9	17.9
Random Forest	7.3	6.2	17.7	16.0
SVR + Linear Kernel	13.0	11.7	20.0	16.9
SVR + Gaussian Kernel	13.5	12.4	18.2	16.2
Baseline				
Historical Mean	27.3	26.2	30.1	28.7

in Table III. Subsequently, we apply each model to the unseen volume pair $S_{\bar{n}}^{(I)}$, $T_{\bar{p}}^{(I)}$, obtaining the corresponding predicted execution time $\hat{t}_{\bar{n},\bar{p}}^{c}$.

We repeat this process first on execution times of the c2 core and then for the f2 core, and using Eq. (2) we estimate the core assignment $c_{\bar{n},\bar{p}}^{*}$ that minimizes cost. For each volume pair, we estimate the time difference between the actual and predicted times $t_{\bar{n},\bar{p}}^{c} - \hat{t}_{\bar{n},\bar{p}}^{c}$, and also record the agreement of assignment, i.e., if $c_{\bar{n},\bar{p}}^{*} = \hat{c}_{\bar{n},\bar{p}}^{*}$. We repeat this process for all 120 volume pairs and calculate the mean absolute percentage error (MAPE) of predicting the execution time for each core service, and the percentage of the 120 pairs for which the assignment to a core service is the same as for the best case.

Using such evaluation criteria allows us to compare the performance of the different models when predicting the execution time of a new registration. We use MAPE since this indicator is widely adopted to determine the quality of estimation, especially in the case of time series forecasts, and allows us to evaluate directly any differences in time prediction between core types. (If we were to use actual times this comparison would not be possible since one core type is faster than the other.)

Given the estimated core assignment $\hat{c}_{\bar{n},\bar{p}}^{*}$, we calculate the average execution time (across all jobs) and the total cost for that allocation. For comparison we use the historical mean of execution times of the training set at each core service as an alternative to the proposed approaches.

Estimating as accurately as possible the execution times of non-linear registration processes is an important aspect of resource provisioning when considering allocation for monetary cost minimization and also for job scheduling. Table IV outlines the main results obtained regarding the execution time estimation and provides a comparison with the historical mean. (SVR with sigmoid and polynomial kernel report low performance in all experiments, and are excluded from the comparison.) Overall, the MAPE in predicting execution time of our learning based approach using a linear model is 17.6% and 15.5% for the c2 and f2 core services respectively. On the other hand, using simply the mean of historical data the error is 30.1% and 28.7% respectively. Thus, our approach provides almost a 50% improvement over current practice. It appears that a linear model

TABLE V: Cross-validated results for resource allocation accuracy.

Model	Training Accuracy	Testing Accuracy		
	Overall[a] (%)	Overall (%)	c2 Allocation[b] (%)	f2 Allocation[b] (%)
Proposed				
Linear Model	83.1	77.5	81.5	69.2
Linear Model + Feature Selection	81.8	73.3	70.4	79.5
Random Forest	92.1	88.3	88.9	87.2
SVR + Linear Kernel	83.8	80.0	82.7	74.4
SVR + Gaussian Kernel	86.1	83.3	84.0	82.1
Baseline				
Historical Mean	62.5	49.2	71.6	2.6

[a] Overall accuracy denotes the percentage of correct allocations with respect to the best case.
[b] c2 (resp. f2) allocation accuracy denotes the percentage of correct c2 (resp. f2) allocations with respect to the best case.

describes well the relationship between the features used and execution time, and offers comparable performance in both training and testing sets, indicating good generalization. The reduced linear model with feature selection (using on average 6 features) does not improve performance, demonstrating the importance of all features (statistical and topological extracted at all resolution levels). The other non-linear models do not seem to provide any benefit in terms of testing performance. While Random Forest does offer the lowest training error ($\approx 7\%$) it does not generalize well, indicating probable over-fitting.

The first direct application of estimating the job execution times is job allocation. When using the PiCloud platform, the central question is to which type of core to allocate each individual job. We are interested in the overall allocation performance (percentage-wise, how many jobs are allocated to the same type of core as in the best case) and the specific allocation performance (percentage-wise, for each core type, the allocation accuracy as compared to the best case). The complete results are depicted in Table V. The linear model agrees with the best case on 77.5% of the cases; whereas using the historical mean to drive decisions results in an overall agreement of only 50%. Remarkably, the historical mean agrees less than 3% in f2 allocations with respect to the best case. Such biased behavior is not observed with the linear model. Overall, all of the proposed models offer superior performance compared to the historical mean, and the Random Forest offers the highest accuracy among all (88.3%). It appears that there is slightly higher accuracy when allocating to c2 rather than f2 core type. However, this is not due to less accuracy of the model (no such difference in execution time prediction were observed before) but because there are fewer allocations to f2 in the best case (32% jobs are allocated to f2, while 68% are allocated to c2). Finally, it appears here that all models perform similarly and sometimes better than the linear model. This is due to the fact that the allocation accuracy is a byproduct of a cost decision. That is, after execution time prediction, the allocation is decided according to cost. Most of the difference in the execution time (estimation error) is masked by the non-linear pricing strategy of real-time cores of PiCloud. On average, as shown in Table II, a registration takes 37 minutes on a f2 core. This is remarkably close to the 60% of the hour (36 minutes) decision threshold for real-time core charge. Thus, what it matters for allocation purposes, at least for f2 core type, is whether both real and predicted execution times are below or above this threshold, and not exactly how accurate the time prediction is. However, if the pricing strategy changes one would expect these accuracy figures to change as well, and this is the compelling reason for reporting prediction time as well, since execution time is the input to any decision problem.

Encouraged by the performance indicators obtained when estimating execution times and allocations, we apply our model to the estimation of the monetary cost of using the PiCloud platform and illustrate the potential monetary gains for intelligently allocating resources. We use the proposed approach to drive core allocations which in turn reduce total monetary cost as seen in the bar plot of Figure 6 (for clarity not all variations are shown). The proposed approaches offer a cost close to the best case scenario. In the same figure the blue line shows the average execution time of each case. Overall, driving decisions using any of the proposed approaches, results on an average execution time of 47.1 minutes on par with the 46.4 minutes of the best case. Thus, adopting any of the proposed approaches one can achieve both lower cost and better performance compared to using the c2 service (longer time), the f2 service (more costly), or the historical mean (more costly and longer time). Naturally, the prediction accuracy will improve by adding more data and a more advanced similarity model (one that is still computationally efficient to estimate).

Several conclusions can be drawn from these findings. The Linear, Random Forest and SVR + Gaussian Kernel models perform very well in terms of time and cost estimation, leading to an approximate 12–15% cost reduction compared with 'All f2' – the most expensive solution. We emphasise that all proposed methods offer quantitatively better results than the historical mean. This leads to the conclusion that inquiry into the content of the input image data has a great importance for execution time estimation and ultimately cost prediction and optimization. This approach seems to provide more insight into the correct allocation patterns than only using data dimension and/or historical information.

In this paper we use a few data points but we can assess how the methods will behave when applied on larger datasets. Assuming the availability of a large number of independent and homogeneous jobs we expect the result of the historical mean to improve significantly. Of course, this setting also implies that the results obtained by the proposed methods will improve as well. Although the difference between the historical mean solution and the proposed solutions may decrease in this case, still we expect the proposed to always outperform the naive historical mean approach and provide a more reliable time estimation mechanism, e.g. for scheduling applications. Conversely, if we consider a more complicated setup (such as more core types, more complex imposed cost function), a more heterogeneous set of imaging volumes, or an application where accurate time prediction is critical (such as the scheduling of large time dependent workflows of imaging applications), the results of the proposed solutions will perform well while we expect a degradation in the results of the historical mean.

IV. Conclusions

In this paper we demonstrate that offloading intensive analyses to the cloud is possible and beneficial, since they are ideally suited to data parallelism (i.e., they are compute-intensive and not limited by inter-process or intra-cloud communication). While the PiCloud platform is convenient and facilitates the use of cloud infrastructures, it is possible to implement similar pipelines using other libraries if the user chooses so. Nevertheless, no matter what environment is used (cloud, cluster, or multi-core

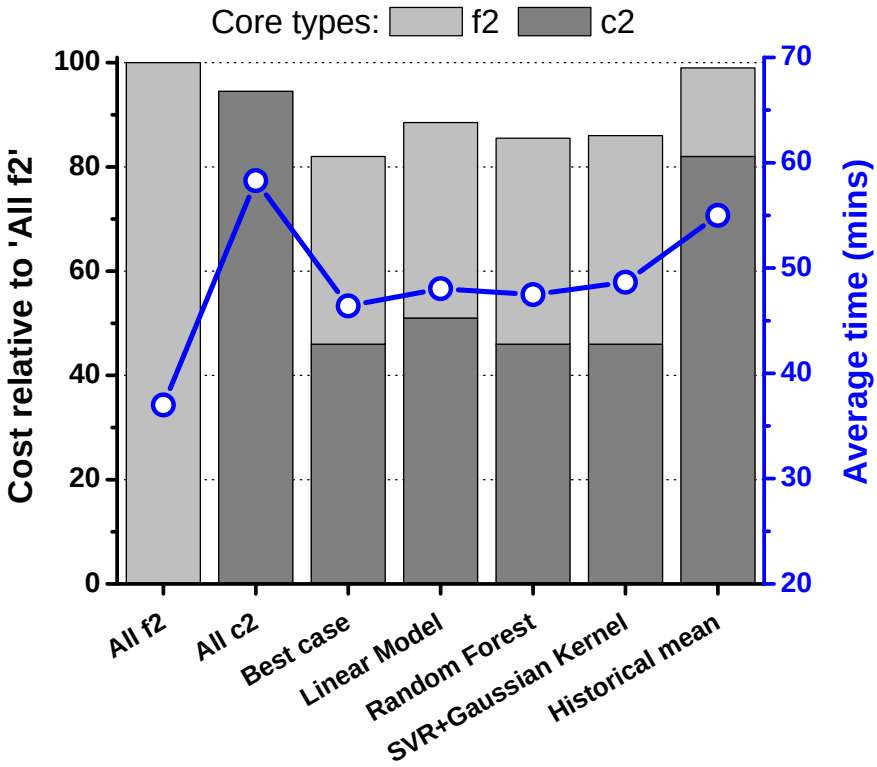


Fig. 6: Cost (relative to assigning all registrations to the f2 core type) and average time of all registrations for various allocation scenarios.

workstations) optimizing resource allocation is necessary and the proposed framework for incorporating data dependency does improve the efficiency of resource allocation methods. While here we demonstrate this potential using simple similarity driven features (statistics of their distribution and topological shape descriptors) for estimating the computational burden of non-linear registrations, more sophisticated methods can be developed. It is clear that the more volumes are registered the more accurate our estimation will be. We are currently developing methods to integrate this prediction process seamlessly, and to extend it to predicting execution time of other core neuroimaging analysis steps (e.g., segmentation). We will release in the public domain all source code material to facilitate adoption from users that are interested in performing such an analysis on the PiCloud platform. Although here we rely on mice data, the same code and approach can be used with any brain MRI data (human or other), and can be easily extended to accommodate different pipelines as well. With this paper our goal is to provide a compelling example of how the cloud paradigm is used to create powerful and comprehensive image analysis pipelines that are simple to use and available to a large community of scientists, thus increasing the democratization of science. The possibility to perform hundreds of parallel executions with a fraction of the cost of owning a workstation opens the possibility to many laboratories around the world to develop state-of-the-art image analysis pipelines, without having to rely only or at all on (shared) distributed computing facilities.

ACKNOWLEDGEMENTS

The authors would like to thank Alberto Galbusera for assistance with data collection and PiCloud Inc., particularly Ken Elkabany, for providing technical assistance.

FUNDING

This work was partially supported by a Marie Curie Action: “Reintegration Grant” [grant number 256534] of the EU’s Seventh Framework Programme (FP7).

REFERENCES

[1] R. A. Heckemann, J. V. Hajnal, P. Aljabar, D. Rueckert, and A. Hammers, "Automatic anatomical brain MRI segmentation combining label propagation and decision fusion," *NeuroImage*, vol. 33, no. 1, pp. 115–126, Oct. 2006.

[2] H. Jia, P.-T. Yap, and D. Shen, "Iterative multi-atlas-based multi-image segmentation with tree-based registration," *NeuroImage*, vol. 59, no. 1, pp. 422–430, Jan. 2012.

[3] A. Klein, J. Andersson, B. A. Ardekani, J. Ashburner, B. Avants, M.-C. C. Chiang, G. E. Christensen, D. L. Collins, J. Gee, P. Hellier, J. H. H. Song, M. Jenkinson, C. Lepage, D. Rueckert, P. Thompson, T. Vercauteren, R. P. Woods, J. J. Mann, and R. V. Parsey, "Evaluation of 14 nonlinear deformation algorithms applied to human brain MRI registration," *NeuroImage*, vol. 46, no. 3, pp. 786–802, Jul. 2009.

[4] G. Antoniu, L. Bougé, B. Thirion, and J.-B. Poline. (2010, Sep.) AzureBrain: Large-scale joint genetic and neuroimaging data analysis on Azure clouds. [Online]. Available: http://www.irisa.fr/kerdata/lib/exe/fetch.php?media=pdf:inria-microsoft.pdf

[5] H. Kim, M. Parashar, D. Foran, and L. Yang, "Investigating the use of autonomic cloudbursts for high-throughput medical image registration," in *10th IEEE/ACM International Conference on Grid Computing*, Oct. 2009, pp. 34–41.

[6] G. B. Frisoni, A. Redolfi, D. Manset, M.-E. Rousseau, A. Toga, and A. C. Evans, "Virtual imaging laboratories for marker discovery in neurodegenerative diseases," *Nature Reviews Neurology*, vol. 7, no. 8, pp. 429–438, Jul. 2011.

[7] I. Dinov, K. Lozev, P. Petrosyan, Z. Liu, P. Eggert, J. Pierce, A. Zamanyan, S. Chakrapani, J. Van Horn, D. S. Parker, R. Magsipoc, K. Leung, B. Gutman, R. Woods, and A. Toga, "Neuroimaging study designs, computational analyses and data provenance using the LONI pipeline," *PLoS ONE*, vol. 5, no. 9, pp. e13 070+, Sep. 2010.

[8] S. Laguitton, D. Rivière, T. Vincent, C. Fischer, D. Geffroy, N. Souedet, I. Denghien, and Y. Cointepas, "Soma-workflow: a unified and simple interface to parallel computing resources," in *MICCAI Workshop on High Performance and Distributed Computing for Medical Imaging*, Sep. 2011.

[9] S. Pandey, D. Karunamoorthy, K. K. Gupta, and R. Buyya, "Megha Workflow Management System for Application Workflows," in *IEEE Science & Engineering Graduate Research Expo*, Melbourne, Australia, 2009.

[10] R. Buyya and S. Venugopal, "The Gridbus toolkit for service oriented grid and utility computing: an overview and status report," in *IEEE International Workshop on Grid Economics and Business Models (GECON)*, Apr. 2004, pp. 19–66.

[11] R. Buyya, S. Pandey, and C. Vecchiola, "Cloudbus Toolkit for Market-Oriented Cloud Computing," in *Cloud Computing*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, vol. 5931, pp. 24–44.

[12] C. Vecchiola, X. Chu, and R. Buyya, "Aneka: a Software Platform for .NET based Cloud Compputing," *High Speed and Large Scale Scientific Computing*, vol. 18, pp. 267–295, 2009.

[13] H. Kim and M. Parashar, "CometCloud: An autonomic cloud engine," in *Cloud Computing: Principles and Paradigms*. Wiley, 2011, ch. 10, pp. 275–297.

[14] K. J. Millman and M. Brett, "Analysis of functional magnetic resonance imaging in Python," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 52–55, May 2007.

[15] M. Minervini, M. Damiano, V. Tucci, A. Bifone, A. Gozzi, and S. A. Tsaftaris, "Mouse neuroimaging phenotyping in the cloud," in *3rd International Conference on Image Processing Theory, Tools and Applications (IPTA)*, Istanbul, Turkey, Oct. 2012, pp. 55–60.

[16] B. B. Avants, C. L. Epstein, M. Grossman, and J. C. Gee, "Symmetric diffeomorphic image registration with cross-correlation: evaluating automated labeling of elderly and neurodegenerative brain," *Medical Image Analysis*, vol. 12, no. 1, pp. 26–41, Feb. 2008.

[17] C. Lin and S. Lu, "Scheduling scientific workflows elastically for cloud computing," in *International Conference on Cloud Computing (CLOUD)*, Jul. 2011, pp. 746–747.

[18] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," in *International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2011, pp. 1–12.

[19] S. K. Garg, R. Buyya, and H. J. Siegel, "Time and cost trade-off management for scheduling parallel applications on utility grids," *Future Generation Computer Systems*, vol. 26, no. 8, pp. 1344–1355, Oct. 2010.

[20] K. Liu, H. Jin, J. Chen, X. Liu, D. Yuan, and Y. Yang, "A Compromised-Time-Cost Scheduling Algorithm in SwinDeW-C for Instance-Intensive Cost-Constrained Workflows on a Cloud Computing Platform," *International Journal of High Performance Computing Applications*, vol. 24, no. 4, pp. 445–456, Nov. 2010.

[21] C. Reynolds, S. Winter, G. Terstyanszky, T. Kiss, P. Greenwell, S. Acs, and P. Kacsuk, "Scientific workflow makespan reduction through cloud augmented desktop grids," in *3rd International Conference on Cloud Computing Technology and Science (CloudCom)*, Nov. 2011, pp. 18–23.

[22] L. Ramakrishnan, J. S. Chase, D. Gannon, D. Nurmi, and R. Wolski, "Deadline-sensitive workflow orchestration without explicit resource control," *Journal of Parallel and Distributed Computing*, vol. 71, no. 3, pp. 343–353, 2011.

[23] E. K. Tabak, B. B. Cambazoglu, and C. Aykanat, "Improving the Performance of Independent Task Assignment Heuristics MinMin, MaxMin and Sufferage," *IEEE Transactions on Parallel and Distributed Systems*, May 2013.

[24] R. Van den Bossche, K. Vanmechelen, and J. Broeckhove, "Cost-Optimal Scheduling in Hybrid IaaS Clouds for Deadline Constrained Workloads," in *3rd International Conference on Cloud Computing (CLOUD)*. IEEE, Jul. 2010, pp. 228–235.

[25] K. Kumar, J. Feng, Y. Nimmagadda, and Y.-H. Lu, "Resource Allocation for Real-Time Tasks Using Cloud Computing," in *20th International Conference on Computer Communications and Networks (ICCCN)*, Aug. 2011, pp. 1–7.

[26] W. Shi and B. Hong, "Resource Allocation with a Budget Constraint for Computing Independent Tasks in the Cloud," in *2nd International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, Dec. 2010, pp. 327–334.

[27] J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente, "Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers," *Future Generation Computer Systems*, vol. 28, no. 2, pp. 358–367, Feb. 2012.

[28] R. Albers, E. Suijs, and P. de With, "Triple-C: Resource-usage prediction for semi-automatic parallelization of groups of dynamic image-processing tasks," in *IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, May 2009, pp. 1–8.

[29] A. Matsunaga and J. Fortes, "On the use of machine learning to predict the time and resources consumed by applications," in *10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, May 2010, pp. 495–504.

[30] W. Smith, I. Foster, and V. Taylor, "Predicting application run times using historical information," in *Job Scheduling Strategies for Parallel Processing*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1998, vol. 1459, pp. 122–142.

[31] N. Kapadia, J. A. B. Fortes, and C. Brodley, "Predictive application-performance modeling in a computational grid environment," in *8th International Symposium on High Performance Distributed Computing*, Aug. 1999, pp. 47–54.

[32] J. Li, X. Ma, K. Singh, M. Schulz, B. De Supinski, and S. McKee, "Machine learning based online performance prediction for runtime parallelization and task scheduling," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, Apr. 2009, pp. 89–100.

[33] L. Shen and F. Makedon, "Spherical mapping for processing of 3D closed surfaces," *Image and Vision Computing*, vol. 24, no. 7, pp. 743–761, Jul. 2006.

[34] E. Bribiesca, "An easy measure of compactness for 2D and 3D shapes," *Pattern Recognition*, vol. 41, no. 2, pp. 543–554, Feb. 2008.

[35] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, Oct. 2001.

[36] H. Drucker, C. J. C. Burges, L. Kaufman, A. J. Smola, and V. Vapnik, "Support vector regression machines," in *Advances in Neural Information Processing Systems 9 (NIPS)*, Dec. 1996, pp. 155–161.

[37] A. J. Smola and B. Schölkopf, "A tutorial on support vector regression," *Statistics and Computing*, vol. 14, no. 3, pp. 199–222, Aug. 2004.

[38] M. Andrle, L. Rebollo-Neira, and E. Sagianos, "Backward-optimized orthogonal matching pursuit approach," *IEEE Signal Processing Letters*, vol. 11, no. 9, pp. 705–708, Aug. 2004.
[39] M. Minervini, C. Rusu, and S. A. Tsaftaris, "Learning computationally efficient approximations of complex image segmentation metrics," in *8th International Symposium on Image and Signal Processing and Analysis (ISPA)*, Trieste, Italy, Sep. 2013, pp. 60–65.
[40] A. E. Dorr, J. P. Lerch, S. Spring, N. Kabani, and R. M. Henkelman, "High resolution three-dimensional brain atlas using an average magnetic resonance image of 40 adult C57Bl/6J mice," *NeuroImage*, vol. 42, no. 1, pp. 60–69, Aug. 2008.
[41] C. C. Chang and C. J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 3, Apr. 2011.